THE JOHN OLIGER CO.

2068 EPROM PROGRAMMER

ERRATA: The following information was left out of the Oliger 2068 Eprom Programmer manual, page 6-7 under "LISTING TWO":

If two 27128s or a 27128 AND a 2764 are required to hold your program, you should use the complete LISTING TWO program to burn the first eprom, but when burning the SECOND eprom you should only use lines 9990-9999 and the routine should be started with (LET p=9990: LET data=9999: LET dest=0: LET one=1: LET three=3) (ENTER) and after the Vpp supply is switched to "Vpp", start as usual with (GO TO p) (ENTER). DO NOT USE LINES 9980-9989 WHEN PROGRAMMING THE SECOND EPROM! The equivalant of this procedure IS covered for 2764 only use in the cartridge board documentation.

THE OLIGER 2068 EPROM PROGRAMMER

CONTENTS:

ASSEMBLY INSTRUCTIONS.....1

USE OF AN ALTERNATE Vpp SUPPLY.....3

GENERAL INFORMATION ON USING PROGRAMMER.....4

PROGRAMMING A 2764 EPROM.....4

PROGRAMMING A 27128 EPROM.....5

OPTIONAL HOME ROM CHANGES.....7

HOW TO INSTALL A 27128 TO REPLACE THE HOME ROM..8

WHERE TO GET THE EPROMS.....9

HOW TO VERIFY YOUR EPROM.....9

STATIC ELECTRICITY PRECAUTIONS.....10

PROGRAMMER THEORY OF OPERATION.....11

PROGRAMMER SCHEMATIC.....12

2068 2764/27128 EPROM PROGRAMMER ASSY INSTRUCTIONS

General Information

Because this pc board features plated through holes, feedthrough wire installation is not required. ALL parts mount on the component side of the board (which is so labeled) and are soldered on the opposite side of the board. Use a clean, low-wattage soldering pencil with a fine tip and very small diameter 60/40 rosin core solder to assemble the board. You will also need: A bottle of acetone & Q-tips or commercial flux remover, a pair of flush cutting "nippy" cutters (Radio Shack) for cutting component leads after soldering, and possibly a small pair of needle nose pliers to form the resistor & cap leads before thier installation.

STEP 1) Install all ic sockets onto the board, making sure that their pin 1 end (the end labeled "1" or having a notch in it) is installed on the same end as the "1" etched onto the board at each ic mounting area. Solder all of these in place, carefully but completely, using extra care at small pads that have pc traces adjacent to them. Extra care here can avoid a lot of wasted time later in having to troubleshoot this board having a tiny short at one of these pads. If a ZIF (zero insertion force) socket is desired for the eprom (supplied by user), it should be installed now instead of the supplied low profile soldertail socket, as removal of these sockets on a plated through board such as this one is very difficult to say the least. This board was laid out in such a way that there is plenty of room available for use by the larger ZIF socket, if one is desired by the builder. After all sockets are soldered in place, remove all traces of flux with acetone and Q-tips or any other commercial flux remover. Carefully inspect your work for solder bridges and bad or incompletely soldered joints. Touch up anything looking at all suspicious.

STEP 2) Install all capacitors & resistors on the board (C1, C2, C3, C4, & R1), at their locations marked on the component side of the pc board. Be sure that C1 is in its correct location as this capacitor is a tighter spec. low drift part. (It is physically larger than the other two .1mf caps supplied in the parts kit) Also be sure that the red (+) end of C4 is installed in the pad labelled (+) on the board.

STEP 3) Install switch SW1 in its correct location on the board. SW1 should be oriented so as to not cover up too much of the "128" legend on the board. Install diode D1 on the board. This diodes band end (cathode) should be installed on the same end as the line on the diodes symbol etched onto the board. This results in the diodes' band end being on the left side of the diode as seen looking down at the board with the boards' edge traces towards you.

STEP 4) Prepare 2 pieces of 22 or 24AWG solid (1=red/1=black) wires, 2 inches long, by stripping 1/4" of insulation from one end of each and 3/4" insulation from the opposite end of each. Bend the longer stripped end of each wire into an "O" with small needle nose pliers. (All of this has already been done to the wires supplied in this boards' parts kit) Install these wires on the board by inserting the shorter stripped end of each wire into its' respective pad on the component side of the board and soldering it on the opposite side of the board. The red wire is inserted into the pad labeled "Vpp" and the black wire is inserted into the pad labeled "Gnd" at the top of the board. The free end of these "Vpp input wires" should be bent to extend above the board (away from the edge traces) for easy attachment of the Vpp supply when programming an eprom.

STEP 5) Clean off all traces of flux from new soldering with acetone or flux remover. Carefully perform a final inspection of your soldering. Touch up your soldering as required. All soldering on this board is now complete!

STEP 6) Install all ics (U1-U5) in their respective sockets per the schematic. Because each different type of ic has a different number of pins, it is very hard to get the ics in the wrong sockets. Be VERY sure, however, that ALL pins of each ic actually enters its' respective contact in the socket & that pin 1 is installed on the same end as the "1" label on the board. (Pin 1 of the ic is on the end of the ic that has a notch or small hole molded into it.) Beware when installing the eprom, later, that its' pin 1 is on the opposite end of the socket in relation to U1-U5, usually resulting in its' part number printing being upside down when the board is installed. Your programmer is now ready for testing!!!

STEP 7) I recommend testing the board before actually trying to program an eprom. With the SW1 in its' "64" position, if you have a logic probe verify that pin 11 of U3-U5 each recieves a pulse when any POKE is made to memory from 8192-16383d. Now key in <LET a=0><ENTER>. Now switch SW1 to its' "128" position and key in <POKE a,a><ENTER> while monitoring pin 11 of U3. You should recieve a pulse on your logic probe. Now use your logic probe or a voltmeter to verify that each used output (see schematic) of U3-U5 is at a logic low. If not, check the board for errors.....it should be. Now flip SW1 to its' "64" position and key in <POKE 16383,255><ENTER>. Use your logic probe or voltmeter again on all of these latched outputs, this time verifying a logic high on each used output of these chips. (Don't be confused by a wrong logic level on a latch output that is not used on one of these chips...there are a couple not used.) If you don't find a logic high on a used output, start checking the board for errors to find out why....you should. This completes testing of this board.

USING YOUR OWN POWER SUPPLY IN LIEU OF THE Vpp SUPPLY

Although the John Oliger Co. Vpp power supply is recommended for use with this programmer, you can also use your own power supply for Vpp if it is a regulated, variable supply capable of covering the voltage range from 4.4 to 21VDC. You will also need a voltmeter to use with it that can cover this range.

To use it with this programmer, you will need to preset it to power up on turn on at about 4.5VDC. Then, when the programming instructions refer to connecting the Vpp supply to the programmer, you connect instead the combination of your preset power supply and voltmeter. When the doc says to plug in the Vpp supply, you should plug in or switch on your own power supply, which should power on at the preset 4.5VDC. When the doc says to switch the Vpp supply to "Vpp", you should at that point carefully increase the voltage on your power supply to 21VDC. Use care here not to overshoot 21V because the eprom cannot handle anything, even for an instant, greater than 22VDC, and any overshoot higher than this WILL destroy your eprom! (Just connecting the power supply preset to 21VDC when ready or switching the Vpp input line with a switch will usually destroy an eprom because of the initial inductive surge/arc!)

When the documentation says to switch the Vpp supply back to 4.4V, you should at that time lower your supply's voltage to about 4.5VDC. And, of course, when the doc says to unplug the Vpp supply you should unplug or switch off your supply.

You will find that there is nothing wrong with this arrangment if you only infrequently program eproms on this board, but are advised, if for nothing else but convenience, to obtain the Oliger Vpp Supply if you find yourself programming eproms a lot!

HOW TO USE THE 2068 EPROM PROGRAMMER

General Information

The 2068 Eprom Programmer is memory mapped @ 8192-16383 with its select (SW1) switch in the "64" position and 0-16383 with SW1 in the "128" position. The programmer will respond only to a write (POKE) in this address space. If the programmer is installed in your expansion board, contains a fresh eprom, and 21VDC is supplied to its Vpp input lines, the programmer will take any data POKEd into it and store it on eprom at the address POKEd for the 27128, or the address POKEd-8192 for the 2764. The programmer contains its own timer and latch circuit and does NOT use the Z80 WAIT NOT control line, so it is compatable with the dynamic memory used in the 2068 computer with no possibility of data corruption. Unlike most eprom programmers, the software required to program an eprom consists of a simple FOR/NEXT loop with a PAUSE 3 statement after each POKE in the loop to allow the timer on the board to finish its programming cycle. Because of the design of this programmer, a verification from this board is not possible. Verification of the data, if desired, must be performed from a 2068 User Cart, and the procedures required will be covered later. It is recommended that eproms with a rated access time of 250ns or less be used on the 2068 microcomputer, although this programmer is cap able of programming any speed of eprom. It has also been my experience to have problems with speed on Intel brand eproms, and I have read of the same problems with Intel eproms used on other programmers and computers (the BBC computer comes to mind immediately), so I recommend avoiding Intel brand eproms for use on the 2068 computer if possible.

PROGRAMMING A 2764 TYPE EPROM

When programming 2764 eproms, use the "64" position of SW1 and POKE your data to locations 8192-16383. The programmers' base address in this mode is 8192, so a POKE to 8192 goes to the eprom's location 0, POKing 8193 goes to the eproms location 1, POKing 8194 goes to location 2, etc. When programming an eprom on this programmer, the eprom can be thought of as that legendary "write only memory", which really is what it is at this point!

Install a new or erased 2764 eprom in the eprom socket of the programmer, with the computer off and programmer removed from the expansion board. Be sure that pin 1 (the end of the eprom with the notch on it) is installed in the socket on the same end as the "1" etched onto the board. Note that the eprom has its pin 1 to the right, while the support chips on the board all have pin 1 to the left. DO NOT get your eprom in backwards as if it is attempted to program one installed this way the eprom will most certainly be destroyed! Now plug the programmer into your 2068 Expansion Board and attach the Vpp supply's output clips to the pgmr boards Vpp input wires. Be sure that the Gnd (black) input wire is connected to the supply's Gnd output and its Vpp (red) input wire connects to the supplys Vpp output.

Turn on your 2068 computer and, after setting your Vpp supply's switches to "4.4" & "21", plug in your Vpp supply. Its green "power on" LED should now light, but its red "Vpp" LED should not light at this time.

You are now ready to program your eprom. To program a 2764 for use on the 2068 User Cartridge, refer to the program and instructions included with your User Cartridge. To program a 2764 with data, such as a mc program, LOAD or key in your code into free ram at this time. After the data desired to be stored on the eprom is in ram, use a FOR/NEXT loop to POKE the data onto the eprom. An example of this follows:

THIS SUBROUTINE WILL TAKE THE CONTENTS OF RAM FROM 32768-40959 (8000-9FFFH) & PUT IT ON A 2764 EPROM. THE PROGRAMMERS SW1 MUST BE SET TO "64" OR ERRORS WILL OCCUR IN THE FIRST FEW LOCATIONS.

Key the program in with the Vpp supply at 4.4V. Flip your Vpp supply to "Vpp" only right before you <GOTO 1>.

```
10 LET x=8192                    ;Base addr of pgmr set to "64"
20 FOR n=32768 TO 40959          ;Get data from 32768 to 40959
30 POKE x,PEEK n : LET x=x+1     ;Pgm 1 byte & point at next one
40 PAUSE 3 : PRINT AT 9,9;x      ;Wait for pgmr to time out and
                                 ;print location just pgmd
50 NEXT n : BEEP 2,2             ;loop for another byte 'til done
                                 ;Beep to signal end if done
```

After the computer is done storing the data on eprom, as known by the report code or the BEEP, switch the Vpp supply back to 4.4V. Now unplug the Vpp supply and then switch off your computer. Remove the programmer board from your expansion board and the eprom from the programmer. Your 2764 is now programmed.

PROGRAMMING A 27128 TYPE EPROM

Programming the 27128 eprom is somewhat more involved than programming 2764s on the TS2068 because the Home Rom contains a routine that likes to write garbage to the first four locations of itself when handling a number from Basic. This usually results in the eproms 4th location (address 0003H) being programmed a zero, despite attempts of other values.

PLEASE NOTE THAT THIS DOES NOT HAPPEN WHEN PROGRAMMING 2764S WITH SW1 IN ITS "64" POSITION, SO NO SPECIAL CARE IN PROGRAMMING IS NEEDED WHEN PROGRAMMING 2764S.

There IS a way around the problem, but care must be exercised nonetheless. The eprom CANNOT be programmed, by yourself or the Home Rom if the Vpp Supply is in its 4.4V position. Also, reading a number from a DATA list or using a variable that has been set up with via a LET statement when Vpp was at 4.4V does not cause the rom to write its garbage. Thus the following two subroutines have been written to program 27128s on this programmer. The first will take data from ram locations 32768 to

49151 & program it on a 27128 eprom. The second subroutine replaces the User Cart documentation routine when programming 27128 eproms with Basic programs for the User Cart. The important things to remember when programming 27128 eproms on the 2068 are:

1) Keep the 4.4/Vpp switch of the Vpp supply in its 4.4 position AT ALL TIMES when entering Basic lines or ANYTHING with numbers from Basic.
2) Switch the Vpp supply to "Vpp" only right before you actually start the program & DON'T start the program with a <GOTO number> statement...you must use a <GOTO variable> statement to start the program, with the variable made equal to the line number desired via a <LET variable=number> statement while the Vpp supply is in its 4.4V setting.
3) Type in the subroutine used (below) EXACTLY as listed, except for the numbers listed in the program as being set for the particular application.
4) Be sure SW1 is in its "128" position.

Other than the care that must be taken into account when programming a 27128, SW1 being in its "128" position, the programmers base address starting at location 0, and the 16K capacity of the 27128 itself, the eprom is programmed just like a 2764. I suggest programming only 2764s until you get the "feel" for the procedure, and then adding on to your knowledge of this programmers use by going to the 27128.

LISTING ONE. This subroutine takes the contents of ram from 32768 to 49151 & programs it onto a 27128 eprom. This data could be mc loaded here from tape, another eprom mapped 32768-49151, or data POKEd into memory in some manner.

```
9990 RESTORE data: READ src: READ end: READ dest: READ one: READ
three: READ loop: READ line: READ col
9992 POKE dest,PEEK src
9993 PAUSE three
9994 LET src=src+one
9995 LET dest=dest+one
9996 PRINT AT line,col;dest
9997 IF src<=end THEN GO TO loop
9999 DATA 32768,49151,0,1,3,9992,10,13  ;First 2 nos can be
                                        ;changed if data in
                                        ;mem is elsewhere
```

BEFORE RUNNING THIS SUBROUTINE KEY IN: <LET p=9990: LET data= 9999><ENTER> then flip SW1 to Vpp and key in <GO TO p> <ENTER>

LISTING TWO. This subroutine takes the place of the one given in the 2068 User Cart documentation when programming 27128 eproms. (PEEK 23627+PEEK 23628*256)-1 cannot exceed 43085 for one 27128 or 59469 for use with two 27128s. The 2nd data item in line 9999 (xxxx) should be the number written down from the User Cart documentation if 43085 or less. If greater than 43085, use 43085

for the first eprom and when programming the second eprom use 43086 to replace 26710 in line 9999 and the number written down to replace the xxxx on this 2nd eprom. Note: If the number was greater than 43085 but no greater than 51277, you can use a 2764 for the 2nd eprom and save some money.

```
9980 RESTORE data: READ dest: READ one: READ three: READ eight
READ next: READ data
9982 READ z: POKE dest,z: LET dest=dest+one
9984 PAUSE three
9986 IF dest<eight THEN GO TO next
9988 DATA 0,1,3,8,9982,9999
9989 DATA 1,2,8,128,15,1,0,0
9990 RESTORE data: READ src: READ end: READ loop: READ line:READ
col
9992 POKE dest,PEEK src
9993 PAUSE three
9994 LET src=src+one
9995 LET dest=dest+one
9996 PRINT AT line,col;dest
9997 IF src<=end THEN GO TO loop
9999 DATA 26710,xxxx,9992,10,13
```

TO START THIS SUBROUTINE, KEY IN: <LET p=9980: LET data=9988> <ENTER> then switch the Vpp supply to its' Vpp setting and key in: <GO TO p><ENTER>

These two programs will get you programming 27128s on the standard TS2068 computer with the supplied home rom. (U16) There IS, however, another way to get around the problem which is much better in the long run if you plan on programming very many 27128 eproms. And what is that? Correct the home rom by transfering it into ram, changing the code, and then programming a 27128 to replace it! How do you change it? Use Hot Z 2068. What are the required changes? They are listed below:

```
002B-002F=84,87,8B,8D,92      ;This is a new little table added
006D 2801      JR Z,0070      ;This is a correction to a NMI bug
37B8 D9        EXX            ;Get exchange registers
37B9 212B00    LD HL,002B     ; Form pointer into
37BC 85        ADD A,L        ; the new table in the
37BD 6F        LD L,A         ; HL register pair
37BE 6E        LD L,(HL)      ;Get LSB of desired const address
37BF 2636      LD H,36        ;MSB of const address is 36H
37C1 D9        EXX            ;Return it in HL'
37C2 AF        XOR A          ;LET A=0 & clear carry flag
37C3 C9        RET            ;Done, so return
37C4 00        NOP            ;Extra byte
```

The above subroutine does the same thing that the routine it replaces did, but it does it without writing garbage to itself

and even does it more efficiently. If you run the benchmark speed tests given in recent issues of Creative Computing both before and after these changes, you will find that the 2068 gains 5 seconds with these changes. Not enough to lie awake at night about but faster nontheless. Also, the changes listed above correct an old error in the NMI handler of the 2068 carried over from the rom in the Sinclair Spectrum. With this change, if you know any mc, you can examine the NMI routine &, I am sure, find out how to use it. (How would you like to add a break key that could stop ANYTHING, including non-breakable programs and run-away machine code!)

Now, getting back to the changes nec. to make these corrections to the home rom. Can you simply mount the new 27128 eprom in the home roms socket without any hardware changes? No, 'fraid not. Can you simply make the trace cuts and wire jumps listed in the TS2068 Technical Manual? You can, BUT if you do your 2068 eprom programmer will no longer work correctly 'cause this will take away the RD NOT decoding required for compatibility with the programmer. So what do you do to mount this eprom in the U16 socket and STILL be able to use the 2068 programmer? You get yourself one 74LS32 or 74HC32 ic and install it inside your 2068 as follows:

Remove all screws from the bottom of the computer case and lift case top from the computer, carefully unplugging the keyboard cable while doing so. Find jumper resistors W1 & W2 near center of board and clip both of these from the board.

Now take your 74LS32 or 74HC32 chip and carefully bend all of its pins straight out at a 90 degree angle from its body. Clip all of this ics leads off where the lead goes from narrow to wide on the pin. Using a small dab of silicone rubber sealant, stick the body of the chip right in middle of the W1-W4 area, leaving access to all four of these pads. Now, using wire wrap wire and small gauge solder, make the following connections to and from this chip. Any pin number not listed is left unconnected.

Pin 1 to left W1 pad. Pin 2 to left W2 pad. Pin 3 to right W1 pad. Pin 14 to right W2 pad AND to U13-pin9. Pin 7 to U12-pin18.

When soldering these wire wrap wires to the wide part of the HC/LS32 chip, solder quickly to avoid damage to the chip. The eprom mod is now done...you can now install your 27128 in the U16 socket. (And you did not even have to make a trace cut!)

If you desire to install a 2764 in the Exrom's socket, (U20) you CAN do it the way Timex says in the Tech manual without any conflicts. Plug back in your keyboard cable and reassemble your 2068 case and we are ready to go.

With these changes made, and the eprom safely inside your TS2068, you no longer have to use those special programs when programming 27128 eproms. You can now use a simple FOR/NEXT loop like shown for the 2764, with the base programmer address changed and the capability of 16K storage instead of 8K.

WHERE TO GET THE EPROMS

One of the best places I know of to get the eproms for use in this programmer is: Microprocessors Unlimited @(918)267-4961.

I also recommend a subscription to: Computer Shopper 407 S. Washington Ave. Titusville, FL 32796. Current subscription price is 1 year (12 issues) for $15.00. This large newspaper type magazine lists many sources for these eproms (prices change on these things constantly, they are now going downhill!) and a full page ad from Micro Unlimited every month. You will also be able to find the best price on printers, monitors, and the like as a lot of discount dealers run ads here. Besides all of this, there is an excellant "Timex-Sinclair Survival Column" by Mark Fendrick run each month...so it really IS a good deal.

HOW TO VERIFY YOUR EPROM IF DESIRED

Because the address space used by this programmer is shared with the 2068 Home rom, and to keep both the hardware & software involved as simple as possible, this programmer does NOT include verify circuitry. To verify an eprom you must either run a checksum on the data to be stored and run the same checksum routine on the eprom after programming, (Hot Z 2068s VERIFY function is good for this on mc) or store the data on tape to be loaded in later for a direct comparison with the programmed eprom in a User Cartridge board. An example of this follows:

You have programmed a 27128 eprom with data that was stored in ram from 49152 to 65535. (the top of memory) You have saved this data to tape with a <SAVE "data" CODE 49152,16384> statement. Install the programmed eprom in the cartridge board, with it mapped 32-48K.

Clear out the top 32K of memory with a <CLEAR 32767> command, then load in the data back to the top of memory with <LOAD "" CODE>. Now key in the following comparison/verification program:

```
10 OUT 244,48
20 LET x=49152:FOR n=32768 TO 49151
30 IF PEEK n<> PEEK x THEN PRINT n;"=";PEEK n,x;"=";PEEK x
40 LET x=x+1: NEXT n
```

If the little program finishes with a clear screen, then the eprom is verified as being 100% exactly like the data. If there is something on the screen, then the screen will be showing you where and what IS on the eprom and what should have been on the eprom.

There is really not much need in trying to verify a Basic program stored on eprom. If the program RUNs correctly without errors then you can be certain it is ok. You will find that the only time a eprom will not verify correctly it will have been because: 1) There was an error made in keying in the burner

program itself. (This usually results in EVERYTHING being wrong) 2) The eprom is defective. (Lots of times a single bit of all locations will not program in one of these) 3) The eprom was not completely erased.

You can verify that an eprom is erased with: (for a 27128)

```
10 OUT 244,48
20 FOR n=32768 TO 49151
30 IF PEEK n<>255 THEN PRINT n
40 NEXT n
```

It is not a bad idea to do this to all erased eproms.

STATIC ELECTRICITY VRS. THE EPROM

All that I can say about static is that it is the no. 1 killer of all eproms. Yes, static can blow that 27128 in no time flat. There is no reason to become "static paranoid" about this matter, however. I recommend the following 2 rules that if strictly adhered to solves 99% of the static problem:

1) Don't even look at your computer or any eproms without first removing your shoes! Socks are ok, but NEVER wear shoes while working on or near your computer.

2) If you need to ship a board containing eproms or other static sensitive MOS parts to myself or anyone else, wrap the entire board in aluminum foil. A lot of static can build up in that white packing material and plastic.

NOTE: It is certainly not necessary to understand the following details on how this circuit works to build and use this programmer. If, when reading the text below, you find yourself "lost", there is no need at all to be concerned. But if you find yourself understanding part or all of it, then so much the better! Generally speaking, the more you understand of the workings of a piece of hardware the more likely you are to be able to use it to its fullest potential.

Nor gate U1 and miniature switch SW1 form an address decoder for the address range from 0-16383 (with SW1 in its "128" position) or from 8192-16383 (with SW1 in its "64" position). If MREQ NOT, WR NOT, A15, & A14 are all logic low, and (when SW1 is in its "64" pos.) A13 is a logic high, an active high signal is generated at U1-pin 5. With SW1 in its' "128" position, A13 is ignored making this high pulse appear regardless of A13's state. (IE. We don't care, in the "128" position, if the memory write is to the 0-8191 chunk or the 8192-16383 chunk)

This active high pulse is applied to both U2-pin 6, and U3, U4, & U5-pin 11. The rising edge of this pulse on U3-U5's pin 11 causes these flip/flops to transfer and hold the current data and address state of their inputs to & on their outputs. These outputs are applied to the eprom being programmed and will remain in this "frozen" state until another write (POKE) to this decoded address space is performed.

Meanwhile, that active high pulse from U1-pin 5 has also been applied to the 555 timer U2 at pin 6. This timer (if you are familiar with the typical wiring of a 555 used as a one shot), is wired somewhat unconventionally so as to respond to an active high trigger pulse and output an active low timed pulse. D1 was added as part of this unconventional wiring, and the timing components (C1 & R1) have been adjusted in value for the correct 50ms pulse because the normally wired 1-shot's equation of 1.1*R*C, for this IC, is no longer accurate.

After the triggering pulse is applied to U2's input at pin 6, there is a typical delay of 100ns before the IC's output at pin 3 goes active low. This delay easily satisfies the eprom's stable address/data line requirements before allowing it's PGM NOT input to go active. This timed, 50ms pulse from U2-pin 3 is applied to the eprom's PGM NOT pin 27.

Now the eprom will take the state of the latched data inputs on its pins 11-19 and store it permanently at the location within itself pointed to by its latched address input lines if:
1) The eprom's Vpp pin has 21VDC applied to it via an external power supply such as the Oliger Vpp Supply set at "Vpp" & "21".
2) The data and address lines are stable for the duration of the 50ms pulse, plus a little more time for the chip to respond to the pulse going inactive. This is accomplished via the latches and the PAUSE 3 Basic statement in the burner program which keeps the loop from POKING this address space for 50ms plus the time used by Basic in executing the program itself.

The programmer is now ready to accept another byte.